一般社団法人　電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
IEICE Technical Report
DE2012-38(2012-12)

オンライントランザクション処理におけるVM挙動の詳細解析

石田　　渉†　横山　大作†　中野美由紀†　豊田　正史†　喜連川　優†

† 東京大学

あらまし　クラウドコンピューティングではeコマースやオンラインゲームなどデータベースをシステムの基盤とするようなアプリケーションでの利用が盛んである。それらのアプリケーションに必要なときに必要なだけの計算資源を提供するために、アプリケーションを構成するサーバ群の負荷分散が重要となっている。これらのシステムでは管理面、コスト面での利点から仮想化技術が利用されており、その負荷分散にはVMライブマイグレーションを利用することができる。本論文では複数のVMをクエリの発行元とするオンライントランザクション処理の負荷分散を考える際に、単純にVMライブマイグレーションを利用しただけではむしろ性能低下を招く可能性があり、データベースのキャッシュまで考慮したVMライブマイグレーションの必要性を示す。またデータベースサーバ間でVMがデータベースのどこを参照しているかの情報を共有することでオンライントランザクション処理の負荷分散による性能低下を防ぐ仕組みを検討する。さらにオンライントランザクション処理におけるVMの挙動を実機を用いて詳細に解析し、オンライントランザクション処理の負荷分散においてVMライブマイグレーションを利用した場合の課題について明らかにする。

キーワード　クラウドコンピューティング, ライブマイグレーション, 伸縮性

Analysis of Virtual Machine during OLTP

Wataru ISHIDA†, Daisaku YOKOYAMA†, Miyuki NAKANO†, Masashi TOYODA†, and Masaru KITSUREGAWA†

† University of Tokyo

Abstract　Cloud Computing is widely used for applications based on databases such as e-commerce or online game. Load distribution is important aspect for such applications to provide computational resources on demand. In these system, virtualization is used from the point of view of managing cost and monetary cost, and VM live migration can be used for load distribution. In this paper, we point out that just using VM live migration in the system where multple VMs issuing queries for load distribution can cause performance degradation, and VM live migration considering databases' caches is necessary. Then we consider the load distribution system prevents performance degradation by utilizing database cache information. Furthermore, we analyze VM behavior during OLTP, and clarify the problem of OLTP load distribution using VM live migration.

Key words　Cloud Computing, Live Migration, Elasticity

1. はじめに

近年 Amazon EC2 [8], Rackspace [10], Microsoft Azure [9], Google Compute Engine [11] といったクラウドサービスが広く利用されている．クラウドコンピューティングはクライアントが必要なときに必要なだけ計算資源を提供することができる新しいIT基盤として認識されている．さらに大きな利点の一つとしてクライアントアプリケーションが意識することなく割り当てる計算資源を動的に増減できるエラスティシティ(伸縮性) の存在が挙げられる．

クラウドコンピューティングでは, eコマースやオンラインゲームなどオンライントランザクション処理が必要なアプリケーションでの利用が盛んである．これらのアプリケーションは利用者数の変動, アクセスされるデータ量の変動が大きくエラスティックな計算資源を提供することがクラウドプロバイダにとって急務となっている．

クラウドコンピューティングにおいてクライアントアプリケーションは仮想化技術を利用してVM(Virtual Machine) として提供されることが一般的である．

仮想化技術を利用した場合, エラスティシティの実現手法と

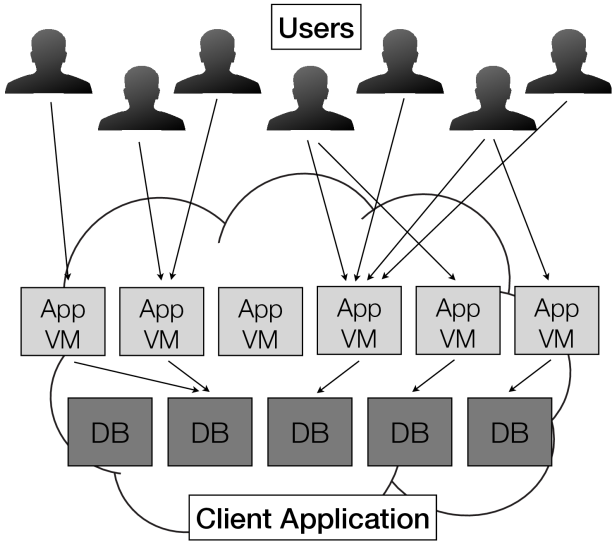

図 1　対象とするクライアントアプリケーションの構成
Fig. 1　client application

して VM ライブマイグレーションが利用できる. 図 1 に本論文が対象とするクライアントアプリケーションの構成を示す. 対象とするアプリケーションは, アプリケーションロジックが実装されるアプリケーションサーバ群とデータ管理を行うデータベースサーバ群から構成され, アプリケーションサーバは VM として実装されているとする.

本論文では大規模なオンライントランザクション処理を行うアプリケーションのエラスティシティを実現するために単純に VM ライブマイグレーションを利用しただけでは, データベースのキャッシュミスによりかえって性能が低下してしまう可能性を指摘する. また VM ライブマイグレーションを行う前にデータベースサーバ間のキャッシュ情報を共有し, 予めキャッシュを温めることでキャッシュミスを減らす仕組みを検討する.

さらにオンライントランザクション処理を行うアプリケーションにおいて VM ライブマイグレーションを行う前後のサーバ群の挙動を実機を用いて詳細に解析し, 大規模データ処理における VM ライブマイグレーションにおける課題について明らかにする.

本論文の構成は以下のようになっている. 2. で現在クラウドコンピューティングで広く利用されている仮想化技術を概説し, 仮想化技術の一要素技術である VM ライブマイグレーションの仕組みを説明する. 3. では本論文が対象とするクライアントアプリケーションのエラスティシティを実現するための課題を挙げ, それに対する解決手法を提案し, 4. で 3. で考察した問題を検証するための計測方法と計測結果を述べる. 次に 5. で関連する研究を紹介し, 最後に 6. で結論を述べる.

2. 仮想化技術と VM ライブマイグレーション

クラウドコンピューティングの実装には仮想化技術が広く利用されている. 仮想化技術は物理マシン上で稼働する HV(Hypervisor) がエミュレートした仮想的な物理環境上で VM を稼働させるもので, VM は自らが仮想化されていることを意識せず, HV を物理マシンと認識し稼働する. クラウドプロバイダは仮想化技術を利用しクライアントに提供する計算資源を VM として扱うことで, 物理的な設定などを必要とせず, クライアントに迅速に計算資源を提供できる. またクライアントはクラウドプロバイダが持つ物理インフラを意識せずに自分専用の計算資源として VM を利用することができる. 代表的な HV の実装として KVM [12] と Xen [13] がある.

VM ライブマイグレーションとは HV 上で稼働している VM を停止させることなく, 異なる HV 上に移す技術で Clark らによって提案された [6]. VM のメモリと CPU ステート, ハードウェアのステートを VM を止めずに段階的にマイグレート先の HV に転送することで行う. VM の動作中はメモリの書き換えが生じるため多くの場合メモリの再転送が必要となり, VM ライブマイグレーションにおいてはこのメモリ転送量の削減がマイグレーションのオーバーヘッドを減らす要となっている. VM ライブマイグレーションにおけるメモリの転送量の削減に関しては多くの研究がなされており, 代表的なものに [3], [4], [5] がある. VM のディスクイメージはマイグレーション元と先の両 HV がアクセスできる仮想ディスクに保管することでマイグレーション後の VM のディスクアクセスを可能にするのが一般的であるが [7] のようにディスクイメージの転送も行うマイグレーションも提案されている. VM ライブマイグレーションにより, クライアントの VM の HV への集約, 分散が可能になり VM が必要とする計算資源と HV が提供できる計算資源のバランスを考慮し VM をライブマイグレーションにより再配置することでエラスティックな計算資源をクライアントに提供することができる.

3. 大規模オンライントランザクション処理を行うクライアントアプリケーションのエラスティシティ

本節では VM として実装されているアプリケーションサーバ群とデータベースサーバ群から構成され大規模なオンライントランザクション処理を行うクライアントアプリケーションのエラスティシティを実現するための課題を考察する.

図 2 に対象とするクライアントアプリケーションの簡単なモデルを示す. アプリケーションサーバを $APPVM1$, $APPVM2$, $APPVM3$, HV を $HV1$, $HV2$, データベースサーバを $DB1$, $DB2$ とする. $DB1$ と $DB2$ はデータベースのレプリケーション技術により同期されているとし, どちらを参照しても同様の結果が得られるとする. 簡略化のためすべてのアプリケーションサーバは $HV1$ 上に集約されて稼働し, $DB1$ を参照しているとする (図 2 ①部).

ここで $APPVM1$ の負荷が上昇したと想定する. その結果 $HV1$ が十分な計算資源を提供できなくなった場合, $APPVM1$ を $HV2$ にライブマイグレートすることによって $APPVM1$ に十分な計算資源を割り当てることができる (図 2 ②部). HV が VM に提供する計算資源には CPU, メモリ, ネットワーク帯域などがある. ライブマイグレーションに伴い $APPVM1$ が必要な計算資源を得ると $DB1$ へのクエリレートが増加する. そう

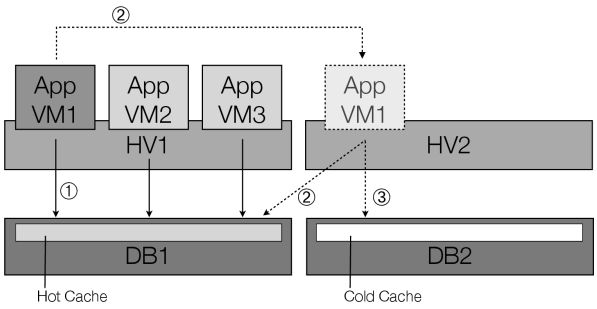

図 2　クライアントアプリケーションモデル
Fig. 2　cloud application model

すると今度は $DB1$ の計算資源が足りなくなりクライアントアプリケーション全体としては負荷の上昇に応えられなくなる状況が発生する可能性がある.

そこで $APPVM1$ のライブマイグレーションと同時に接続するデータベースサーバを $DB1$ から $DB2$ に切り替えることでデータベースサーバの負荷も分散することを考える (図 2 ③部).

しかしこの場合でも $APPVM1$ の上昇したクエリレートにデータベースサーバが応えられない可能性がある. それは $DB1$ が $APPVM1$ からの参照の際利用していたデータベースキャッシュを $DB2$ は持っていないためである.

アクセスするデータがデータベースキャッシュにある場合, データベースサーバからのデータの読み出しはメモリアクセスで済むのに対し, データベースキャッシュにない場合はディスクアクセスを行わなければいけない.

そのためクライアントアプリケーションが大規模なオンライントランザクション処理を行う場合, 接続するデータベースサーバを変更するとキャッシュが温まるまでディスクアクセスが頻発することがクライアントアプリケーションにとって大きなオーバーヘッドになってしまうと考えられる.

以上がクラウドコンピューティングにおいて大規模オンライントランザクション処理を行うクライアントアプリケーションのエラスティシティがアプリケーションサーバの VM ライブマイグレーションだけでは達成できない理由である.

上述した問題を解決するにはデータベースサーバ間でアプリケーションサーバが参照するデータの情報を共有し, 事前に接続先の変更に備えて各データベースキャッシュ上にアプリケーションサーバが参照するデータをプレロードしておけばいい.

しかしデータベースサーバのメモリも有限であり, アプリケーションサーバからのクエリを処理しきるのに必要のないデータベースサーバのディスクにまでデータをプレロードすることは省電力化の観点からも好ましくないため, プレロードは必要最小限に抑えなければいけない.

また実際の環境では各データベースサーバはすでにいくつかのアプリケーションサーバと接続しており, アプリケーションサーバ毎に参照するデータ分布に偏りがあると考えられる. このとき各データベースのデータベースキャッシュにのっているデータは各々異ってくるため, あるアプリケーションサーバの接続するデータベースサーバの負荷が大きくなり, 接続先を変更しなければならない場合, アプリケーションサーバの参照するデータ分布に近いデータがすでにキャッシュに存在するデータベースサーバに接続すればプレロードのコストを削減することができる.

よって単純にアプリケーションサーバが参照するデータを各データベースサーバがプレロードするのでは不十分であり, アプリケーションサーバの負荷変動を監視し, ライブマイグレーションが必要となりそうなアプリケーションサーバを探知し, ライブマイグレーションの結果接続しているデータベースサーバへの負荷が上昇しその負荷に耐えられなそうな場合は, そのアプリケーションサーバの参照するデータ分布から考えて最適な接続先に変更, プレロードが必要な場合はそのデータベースサーバにプレロードを促すような包括的にクライアントアプリケーションのエラスティシティを実現していく仕組みが必要と考えられる.

クラウドコンピューティングにおけるデータベースのエラスティシティに関する研究としては Aaron らによる Albatross [2] や Sudipto らによる Zephyr [1] があるが, 両者はデータベースサーバ自身のライブマイグレーションを実現するものでアプリケーションサーバとの協調は考えられていない. Albatross や Zephyr のアプローチはアプリケーションサーバ群とデータベースサーバ群とのアイソレーションを保ったままエラスティシティを実現しようとするものなのに対し, ここで検討した手法はデータベースがクライアントアプリケーション全体のエラスティシティに関してイニシアチブを持つことを想定している. 具体的な手法とその実装方法は今後の課題とし, 次節では本節で指摘したキャッシュミスによる性能低下が実際に生じるのかを検証する.

4. 大規模オンライントランザクション処理における VM 挙動の解析

前節でアプリケーションサーバの VM ライブマイグレーションを行い, アプリケーションサーバの接続するデータベースサーバを変更した場合, データベースの接続先を変更する前に利用していたデータベースキャッシュが利用できないことがクライアントアプリケーションにとってオーバーヘッドになる可能性を説明した. 本節では実機を用い接続するデータベースの変更がクライアントアプリケーションにとって実際にオーバーヘッドになるのかを検証する.

4.1 計測環境

クライアントアプリケーションとして OLTP ベンチマークである TPC-C [14] を一部改変して利用した. TPC-C は注文, 支払, 注文確認, 発送, 在庫確認の 5 種類のクエリをデータベースに対して発行し, 注文クエリが 1 分間に何回処理できるか (TPMC) で評価を行う. 5 種のクエリのうち, 注文, 支払, 発送は更新系のクエリ, 注文確認, 在庫確認は参照系のクエリであるが, TPC-C の場合, 規定値では更新系のクエリが約 95%となっている. データベースにおけるレプリケーションでは参照系のクエリの負荷分散では有効に働くが, 更新系のクエリはすべて

表 1　計算機環境
Table1　Conputational Environment

HV	CPU	AMD Opteron 6100 2. 2GHz
	memory	512GB
	OS	Debian 6. 0. 4 Squeeze
	kernel	Linux 2. 6. 32
	qemu	qemu-kvm-1. 0. 1
	network	Intel 82599EB 10Gbit
VM	CPU	1 virtual CPU
	memory	1GB
	OS	Ubuntu 11. 04
	kernel	Linux 3. 0. 0
	blk driver	virtio-blk
	network	virtio-net
DB	CPU	Intel Xeon E5530 2. 4Ghz
	memory	24GB
	OS	Debian 6. 0. 5 Squeeze
	kernel	Linux 2. 6. 32
	DB	MySQL 5.5
	network	Intel 82599EB 10Gbit

のデータベースサーバで処理され, 負荷分散されない. そのため TPC-C を規定値通り実行した場合, データベースのレプリケーションによる負荷分散の効果を期待できない. そこで本論文の想定するデータベースのレプリケーションが効果を持つアプリケーションを模擬するため TPC-C の 5 種類のクエリのうち参照系のクエリのみを実行するように規定値を変更した.

変更した TPC-C クライアントを次に説明する各状況下で VM 内で実行し, 注文確認クエリを毎分何回処理できるか (TPM) を各 VM 毎に計測した. 起動する VM は 2 台から 24 台の範囲で変化させ, TPC-C のパラメータである warehouse は 10, connection は 8, 実行時間は 400 秒とした. 計算機環境は表 1 にしめす。

次に計測を行った 7 種類の状況の説明を行う。各状況の概略図を図 3 にしめす。

(a)　全 VM を $HV1$ 上で稼働させた。各 VM は $HV1$ の物理 CPU コアを 1 個占有するよう設定した. VM が稼働する物理 CPU コアの設定には Linux の CPU affinity の機能を用いた. 全 VM は $DB1$ に接続し, $DB1$ のキャッシュは予め温め計測を行った.

(b)　全 VM を $HV1$ 上で稼働させた。各 VM は $HV1$ の物理 CPU コアを 2VM で 1 個共有するよう設定した. 全 VM は $DB1$ に接続し, $DB1$ のキャッシュは予め温め計測を行った.

(c)　全 VM を $HV1$ 上で稼働させた。各 VM は $HV1$ の物理 CPU コアを 1 個占有するよう設定した. 全 VM のうち半分を $DB1$, もう半分を $DB2$ に接続し, 両者のキャッシュは予め温め計測を行った.

(d)　全 VM を $HV1$ 上で稼働させた。各 VM は $HV1$ の物理 CPU コアを 2VM で 1 個共有するよう設定した. 全 VM のうち半分を $DB1$, もう半分を $DB2$ に接続し, 両者のキャッシュは予め温め計測を行った.

(e)　状況 (a) から半数の VM を $HV2$ にライブマイグレートした. 各 VM は稼働する HV の物理 CPU コアを 1 個占有するよう設定した. またすべての VM は $DB1$ に接続し, $DB1$ のキャッシュは予め温め計測を行った.

(f)　状況 (a) から半数の VM を $HV2$ にライブマイグレートした. 各 VM は稼働する HV の物理 CPU コアを 1 個占有するよう設定した. 全 VM のうち半分を $DB1$, もう半分を $DB2$ に接続し, 両者のキャッシュは予め温め計測を行った.

(g)　状況 (f) で温まっていた $DB2$ のキャッシュをクリアした状態で計測を行った.

以降この 7 種類の状況下での計測をそれぞれ計測 (a), (b), (c), (d), (e), (f), (g) と呼ぶ.

4.2　計測結果

図 4 に計測 (a) から (g) の計測結果を示す. 横軸は TPC-C クライアントを実行させた VM の台数, 縦軸は全 VM の TPM の合計値となっている. また図 6 は計測 (g) において $DB1$ に接続する VM の TPM の合計値と $DB2$ に接続する VM の TPM の合計値を分けて表したものである. まず図 4 から VM の台数に比例して TPM の合計値がスケールしているのは計測 (c), (d), (f), (g) となり, 計測 (a), (b), (e) は VM の台数が 12 台を超えた付近から TPM の合計値が頭打ちとなることがわかる. 計測 (a), (e) を比較すると計測 (e) では全 VM のうち半数を $HV2$ にライブマイグレートしているにもかかわらず, 計測 (a) と結果にほとんど差がない. また $DB1$ のみにクエリが集中する計測 (a), (e) と $DB1$ と $DB2$ に分散される計測 (b) を比較すると計測 (b) のほうが VM の台数が 12 台を超えた付近での合計 TPM が約 7%高いため計測 (a), (e) でのボトルネックはアクセスの集中するデータベースサーバになっていることがわかる. しかし計測 (b) も VM の台数が 12 台を超えるとスケールすることができておらず, 計測 (f) で HV を $HV1$, $HV2$ の 2 台用いて負荷分散することでこれを解決できているため計測 (b) のボトルネックは HV になっていることがわかる. 計測 (b) において各 VM は HV の物理 CPU コアを 1 つ占有しているため, ボトルネックが HV の CPU であるなら VM の台数を増やしても計測 (c), (d) のように合計スループットは増加するはずである. また表 1 にあるように VM のメモリが 1G に対して HV のメモリは 512G, ネットワーク帯域は 10Gbit で計測により帯域を使い切っていないことがわかったため, ボトルネックとなっているのは多量の小さいパケットの処理によるものと考えられる.

計測 (c), (d) はスケールはしているものの, 全体を通してスループットは低い. これは 1 個の HV の物理 CPU コアを 2 台の VM で共有しているために VM が十分な CPU 資源を得られていないためと考えられる. 計測 (d) においてクエリを $DB1$ と $DB2$ に分散しても計測 (c) と結果が変化しないため計測 (c), (d) のボトルネックは HV になっていることがわかる.

計測 (f) は結果を評価するために次の簡単な計測を行った. $HV1$ 上で稼働する 1 台の VM をキャッシュの温まった $DB1$ に接続させ, TPC-C クライアントを実行した. その結果 TPM は 2476 となった. 計測 (f) の結果から全ての VM の平均 TPM を

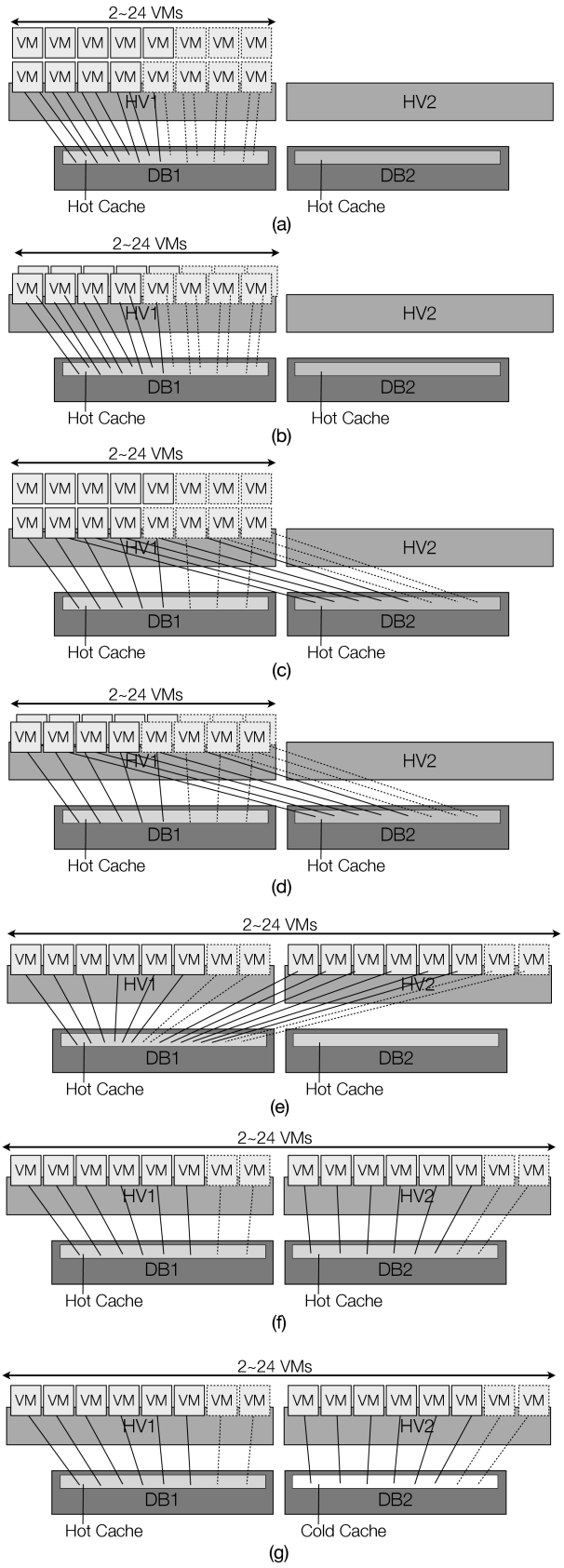

図 3　計測概略図

Fig. 3　Experiment Environment

算出すると最低でもベースラインの 87%の値は得られている．平均 TPM とベースラインを図 5 にしめす．

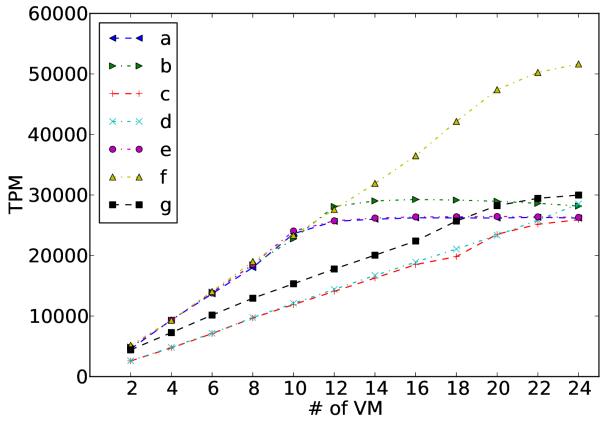

図 4　計 測 結 果

Fig. 4　Experiment result

次に計測 (g) もスケールはしているものの, 計測 (f) と比較するとスループットは低い．図 6 よりこれは $DB2$ のキャッシュが温まっていないために $DB2$ に接続した VM の TPM が $DB1$ に接続している VM に比べ著しく低いためであることがわかる．

以上の計測結果からクラウドコンピューティングにおいて大規模なオンライントランザクション処理を行うクライアントアプリケーションでは HV, データベースともにボトルネックとなる可能性があり, VM ライブマイグレーション, 接続先データベースの変更により一部解決できることがわかった．しかしまた 3. で指摘した通り, VM ライブマイグレーション, 接続先データベースの変更を行なっても, 変更先のデータベースのキャッシュが温まっていないためにむしろ性能が低下してしまう可能性があることも計測から検証できた．クライアントアプリケーションの構成サーバのうち, どこがボトルネックとなるかこの測定では複数の状況での計測を行ったために見当をつけることができた．しかし実際にはクラウドプロバイダはクライアントとの間で結ばれた SLA を満たさなければいけないため, クライアントアプリケーションの負荷が上昇し, 必要なスループット, レスポンスタイムが得られない場合, その時点でどこがボトルネックとなっているのかを判断しなければいけない．3. で検討した手法においてもクライアントアプリケーションの必要とする計算資源を提供しきれなくなった時, アプリケーションサーバをライブマイグレートすればいいのか, 接続するデータベースを変更すればいいのか, あるいは両方必要なのかを判断する仕組みも併せて考える必要がある．

5. 関 連 研 究

クラウドコンピューティングにおけるデータベースのエラスティシティに関する研究としては 3. でも紹介した通り Sudipto らによる Zephyr [1] と Aaron らによる Albatross [2] がある．

Zephyr はそれぞれ独立した契約ポリシーを持つ複数クライアントに物理インフラを共有させるマルチテナントなデータベースに対して, エラスティックな負荷分散を行うためのデータベースのライブマイグレーション手法である．Zephyr ではマイ

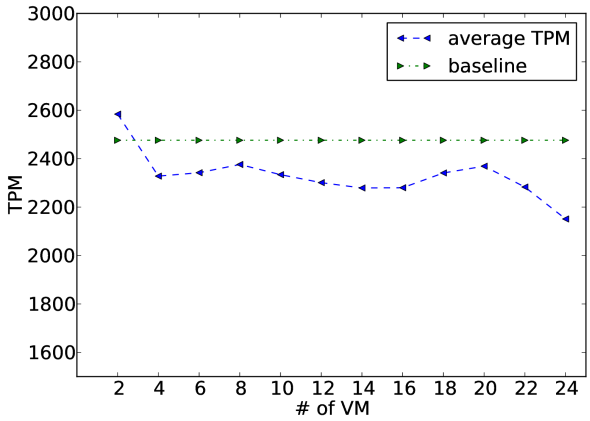

図 5　計測 (f) - 平均 TPM とベースライン
Fig. 5　Experiment (f) -average TPM and baseline

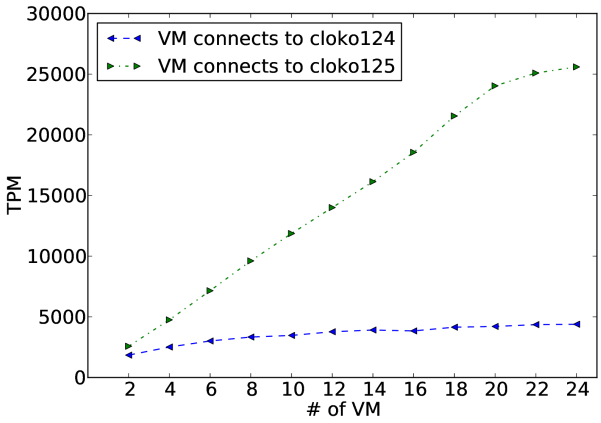

図 6　計測 (g) - 接続先 DB による違い
Fig. 6　Experiment (g) - Differences between connected DB

グレーション元とマイグレーション先はデータベースのテーブルを含め何も共有していない状況 (Shared Nothing) を前提としており, ライブマイグレーションの前後を通してトランザクション処理の中断を生じさせないためにライブマイグレーションが始まる時点で既に実行が始まっているトランザクションはそのままマイグレーション元で行い, ライブマイグレーション中に新たに発生したトランザクション処理はマイグレーション先で行う. マイグレーション中はトランザクションを完遂させるのに必要なデータ揃っていないため, そのようなデータはオンデマンドにマイグレーション元から読み込む手法を提案している. Albatross も Zephyr と同様マルチテナントなデータベースのライブマイグレーションであるが, Zephyr とは異なり各物理ノードがストレージを共有しているシステムを前提としている. Albatross では VM ライブマイグレーションのメモリの転送と同様, トランザクションテーブルをイテレーティブに転送することでライブマイグレーションの前後を通してトランザクション処理の中断を避ける.

6. ま と め

本論文ではクラウドコンピューティングにおいて利用が盛んな大規模なオンライントランザクション処理を行うデータベースを基盤とするクライアントアプリケーションのエラスティシティを実現するための課題を考察し, 仮想化環境でエラスティシティを実現するために利用できる VM のライブマイグレーションを単純に利用するだけではむしろ性能低下を引き起こす可能性があることを指摘し, 問題を解決するための手法を検討した. また実際に想定する環境を構築し, 指摘した問題が生じうるものであることを確かめた. 今後は今回得られた知見を元に検討手法の詳細とその実装方法を考えていきたい.

謝辞　本研究の一部は、総務省委託研究「広域災害対応型クラウド基盤構築に向けた研究開発（高信頼クラウドサービス制御基盤技術）」において実施された。

文　献

[1] Aaron J.Elmore, Sudipto Das, Divyakant Agrawal, Amr El Abbadi *Zephyr: Live Migration in Shared Nothing Databases for Elastic Cloud Platforms* SIGMOD, 2011.
[2] Sudipto Das, Shoji Nishimura, Divyakant Agrawal, Amr El Abbadi *Albatross: Lightweight Elasticity in Shared Storage Databases for the Cloud using Live Data Migration* VLDB, 2011.
[3] Jie Zheng, T.S.Eugene Ng and Kunwadee Sripanidkulchai. *Workload-Aware Live Storage Migration for Clouds.* VEE, March 2011.
[4] Samer Al-Kiswany, Dinesh Subhraveti, Prasenjit Sarkar and Matei Ripeanu. *VMFlock: Virtual Machine Co-Migration for the Cloud* HPDC, June 2011.
[5] Umesh Deshpande, Xiaoshuang Wang and Kartik Gopalan. *Live Gang Migration of Virtual Machines* HPDC, June 2011.
[6] C. Clark, K. Fraser, Steven Hand, et al. *Live Migration of Virtual Machines* NSDI, 2005.
[7] T. Hirofuchi, H. Ogawa, H. Nakada, et al. *A Live Storage Migration Mechanism over WAN for Relocatable Virtual Machine Services on Clouds* CCGrid, 2009.
[8] Amazon Elastic Compute Cloud(Amazon EC2). http://aws.amazon.com/ec2/.
[9] Microsoft azure. http://www.microsoft.com/azure
[10] rackspace http://www.rackspace.com/
[11] Google Compute Engine. https://cloud.google.com/products/com engine
[12] KVM http://www.linux-kvm.org/page/Main_Page
[13] xen http://xen.org/
[14] TPC-C http://www.tpc.org/tpcc/